Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Šilc

Orodje za analizo in testiranje komunikacijskih protokolov

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Mojca Ciglarič

Ljubljana 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Naredite kratek pregled možnih načinov abstrakcije komunikacijskih protokolov s stanji (stateful) in načinov formalne analize pravilnosti komunikacijskih protokolov. Izberite preprost, vendar dovolj močan formalizem, ki bo primeren za uporabo v študijskem procesu na FRI. Načrtujte in implementirajte orodje za analizo protokolov kot spletno aplikacijo. Izberite standardne tehnologije in utemeljite njihov izbiro. Orodje predstavite in demostrirajte njegovo uporabo na resničnih protokolih. Izvedbo kritično ovrednotite in predlagajte možnosti za nadaljnje delo.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nejc Šilc z vpisno številko 63110325 sem avtor diplomskega dela z naslovom:

Orodje za analizo in testiranje komunikacijskih protokolov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) in ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 5. februarja 2016 Podpis avtorja:

Za izdelavo diplomskega dela se zahvaljujem družini, prijateljem in sošolcem, saj so prav oni tisti, ki so me vsa leta študija spodbujali, bodrili in mi polepšali še tako slab dan. Brez njih konec nebi bil tako sladek. Rad pa bi se zahvalil tudi mentorici dr. Mojci Ciglarič za dodelitev zanimive teme, sodelovanje in pomoč pri končni izdelavi.

Don't wish it were easier, wish you were better.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
KP	Communication protocol	Komunikacijski Protokol
LPA	Logical Protocol Analyzer	Logični Protokolni Analizator
PGSS	Perturbation of global state system	Pertrubiranja Globalnega Stanja Sistema
FSM	Final State Machine	Končni avtomat stanj
MVC	Model View Controller	Model Pogled Kontroler

Povzetek

V diplomskem delu so predstavljene različne metode analiziranja in testiranja komunikacijskih protokolov. Preučil sem jih z namenom, da izberem metodo, ki je dovolj preprosta in obenem dovolj učinkovita za implementacijo in uporabo za študente FRI. Metodo sem nato tudi implementiral kot spletno aplikacijo, ki omogoča formalen opis protokola in izvedbo analize logične pravilnosti. V nadaljevanju sem opisal kako je implementacija potekala. Izbral sem primerne tehnologije in utemeljil njihovo primernost. Veliko časa sem namenil testiranju, zato sem postopek in rezultate testiranja nekaterih protokolov opisal tudi v diplomskem delu. Predstavil sem testiranje preprostega protokola s tremi stanji in enega kompleksnega realnega protokola. Kot realni protokol sem izbral protokol TCP, testiral pa sem le del funkcionalnosti, in sicer vzpostavljanje in rušenje povezave. Za konec sem svoje ustvarjanje še pokomentiral in dodal nekaj predlogov izboljšav, ki jih bom poskušal uresničiti v prihodnosti. Aplikacija je nameščena na Heroku, objavljena na GitHubu (https://github.com/nejcsilc/lpa), za uporabnike pa dosegljiva na naslovu http://lpa3.site.

Ključne besede: pgss, lpa, analiza, test, protokol, komunikacija.

Abstract

This thesis presents different methods of analyzing and testing of communication protocols. I have studied them in order to choose the method that is simple enough and at the same time sufficiently effective to implement and use for students of FRI. I have also implemented the method as a web application that allows a formal description of protocol analysis and implementation of logical correctness. This is fallowed by describing of how the implementation was carried out. I have chosen appropriate technologies and justified their suitability. Much time was spent for testing, so in this thesis I have described the process and the results of the testing of some protocols. I have introduced a simple test protocol with three states and a complex real protocol. As a real protocol I have chosen TCP, but I have only tested the establishment and the termination. In conclusion I have commented my creation and added some suggestions for improvements, which I will try to achieve in the future. The application is deployed on Heroku, published on GitHub (https://github.com/nejcsilc/lpa), and available for users at http://lpa3.site web address.

Keywords: pgss, lpa, analysis, test, protocol, communication.

Poglavje 1

Uvod

Analiziranje in testiranje komunikacijskih protokolov[3] se je začelo s prvim povezovanjem računalniških sistemov v računalniške mreže, to je bilo že v začetku sedemdesetih let. Sprva je bilo to področje zlasti zanimivo za raziskovalce in raziskovalne skupine, ki so postavili trdno teoretično osnovo in dosegli, da so predlagani koncepti in rešitve hitro prodrle do uporabnikov ter implementatorjev protokolov. Kmalu so prišle na voljo razne rešitve, ki so pripomogle k hitri analizi novo nastalih komunikacijskih protokolov. Tudi danes se velikokrat poraja želja po novi implementaciji namenskega protokola, ki bi reševal specifičen problem in je implementiran s strani manjše skupine programerjev oziroma načrtovalcev. To pomeni, da potrebujejo programsko opremo za hitro in dokaj enostavno analiziranje takega komunikacijskega protokola, ki pa je hkrati lahko zelo kompleksen. V nadaljevanju je predstavljena implementacija aplikacije, ki naj bi služila prav temu namenu.

Dostopnost aplikacije je enostavna in brezplačna, za uporabo ni potrebna predhodna namestitev, uporabniki pa do nje lahko dostopajo neodvisno od operacijskega sistema oziroma naprave. Ker je namenjena širši množici oziroma uporabnikom po celem svetu, med drugim omogoča tudi večjezičnost. Komunikacijskih protokolov, ki jih uporabniki načrtujejo, ni treba shranjevati na lokalni disk, saj se shranjujejo v skupno podatkovno zbirko. Aplikacija je namenjena tudi zelo zahtevnim uporabnikom oziroma načrtovalcem kompleksnejših komunikacijskih protokolov.

Zaradi preglednosti in shranjevanja komunikacijskih protokolov v skupno podatkovno zbirko aplikacija zahteva prijavo uporabnika. V aplikacijo se je na

začetku treba registrirati. Za še preprostejšo uporabo pa omogočena tudi prijavo z obstoječim Google računom.

Pričakovanja aplikacije so bila zahtevna, vendar nam današnje tehnologije zelo poenostavijo implementacijo, več o tem v poglavju 3. Za implementacijo je bilo treba izbrati še postopek oziroma algoritem, ki omogoča enostavno in hitro analiziranje komunikacijskega protokola, hkrati pa nam da dovolj povratnih informacij, ki nam sporočajo ali je naš protokol ustrezen ali ne, več v nadaljevanju.

Poglavje 2

Analiza komunikacijskih protokolov

2.1 Uvod

Za analizo [2] komunikacijskih protokolov je bilo do danes razvitih kar nekaj postopkov [8], ki uspešno rešujejo dane probleme. Pri večini gre za manjše implementacije raznih skupin oziroma inštitutov, ki pa odkrivajo le delne napake. Seveda so se pojavile tudi naprednejše in kompleksne rešitve velikih korporacij, kot je na primer IBM, ki pa so neprimerne za vsakdanjo oziroma hitro uporabo. Leta 1962 je prišla na voljo še ena zanimiva rešitev, in sicer Petrijeve mreže[5], ki se je do danes razvila v zelo uporabno in pokriva veliko področje, ni pa namenjena samo za analizo komunikacijskih protokolov. Vse do zdaj omenjene rešitve se nanašajo na deterministične metode. Pojavile so se tudi metode z uporabo hevristike, vendar imajo prve še vedno boljše in za prakso sprejemljivejše rezultate.

Preden začnemo bolj podrobno spoznavati tehnike analiziranja komunikacijskih protokolov, pa je treba razumeti, kaj pomeni oziroma kaj si predstavljamo pod pojmom komunikacijski protokol. Kot je omenjeno v knjigi Toneta Vidmarja, Analiza porazdeljenih sistemov [9], je komunikacijski protokol predpis (opis, standard), ki v kontekstu komunikacijskega sistema opredeljuje način komuniciranja med porazdeljenimi procesi (sistemi, aplikacijami). Komunicranje v tem kontekstu

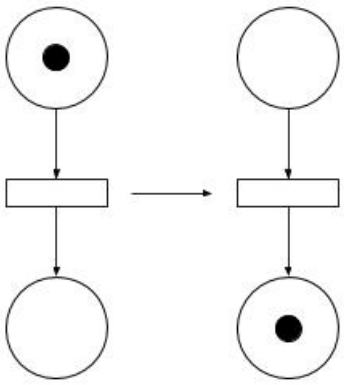
pomeni medsebojno izmenjevanje komunikacijskih sporočil.

2.2 Petrijeve mreže

Temelje Petrijevih mrež je leta 1962 postavil Rus Carl Adam Petri [1]. Postavil je definicijo osnovnih Petrijevih mrež. Kot sem že omenil, so se do danes močno razširile in tako nastale tudi razne različice. Mednje sodijo barvne, časovne, stohastične in mehke Petrijeve mreže. Predvsem prve, barvne, se uporabljajo za analiziranje komunikacijskih protokolov. Na začetku modeliramo oziroma predstavimo KP kot dinamičen sistem, nato poženemo simulacijo in tako pridemo do analize dinamike, ki nam pove ali je slednja v sistemu ustrezna ali ne. Pod pojmom dinamike v sistemu smatramo časovno sekvenco stanj sistema. Dinamičen sistem torej skozi čas spreminja svoja stanja. Analiza dinamike nam pomaga pri odločitvah, ali je sistem treba popraviti, dopolniti itd.

2.2.1 Osnovne Petrijeve mreže

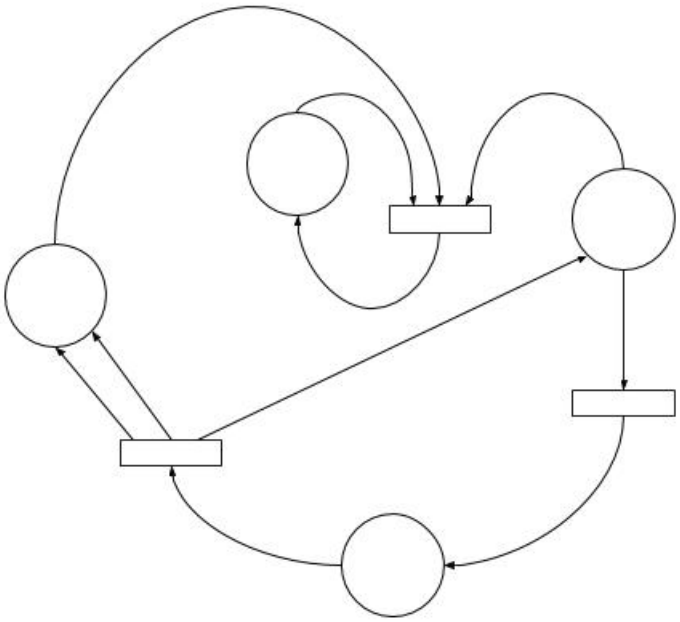
Dinamičen sistem je predstavljen kot bipartitni usmerjen graf, kjer vozlišča prestavljajo akcije in pogoje, povezave pa predstavljajo prehode. Graf vsebuje tudi žetone s katerimi si pomagamo oziroma pogojujemo, ali se akcija lahko izvede ali ne.



Slika 2.1: Ponazoritev izvedbe akcije v grafu

Leva stran slike 2.1 prikazuje dva pogoja in akcijo med njima. Da se taka akcija lahko izvede, v pogoju potrebujemo toliko žetonov, kolikor je povezav iz pogoja do akcije. Po izvedbi akcije se zgenerirajo novi žetoni, in sicer toliko, kolikor je povezav iz akcije v ostale pogoje. Stanje po izvedbi akcije prikazuje desna stran slike.

Na sliki 2.2 je predstavljen celoten primer grafa Petrijeve mreže. Prikazano je, da neko akcijo lahko pogojujemo z več pogoji in da tudi povezave iz akcije lahko vodijo v več drugih pogojev ali pa celo nazaj v isti pogoj.



Slika 2.2: Primer grafa Petrijeve mreže

Da tak sistem oziroma simulacija sistema ustrezno steče, potrebujemo inicializacijski vektor, ki nam določa začetno število žetonov v pogojih. Lahko pa uporabimo tudi generator žetonov, to pomeni, da ob vsaki časovni sekvenci generiramo en nov žeton ali več.

Iz proženja akcij in prehajanja žetonov lahko naredimo analizo, in sicer zgradimo drevo, na katerem prikažemo, koliko žetonov je v posameznih pogojih ob vsakem koraku. Korak predstavlja sprožitev ene akcije ali več. Po končani analizi pridemo do rezultatov. Ugotovimo lahko, ali se v sistemu nabira preveč žetonov, kar pomeni neskončne zanke oziroma cikle. Lahko pa ugotovimo tudi, da se nekatere akcije ne izvedejo, saj se pogoji nikoli ne izponijo, kar označujemo kot mrtvo kodo.

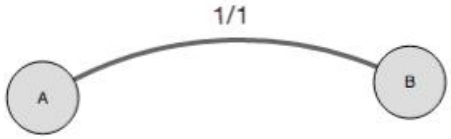
2.2.2 Barvne Petrijeve mreže

Kot sem že omenil, so barvne Petrijeve mreže nadgradnja osnovnih. Razlika je v žetonu, ki zdaj dobi večji pomen. Slednji ne predstavlja samo pogoja za izvršitev

akcije, ampak nosi dodatne informacije (sporočila), ki se prenašajo po sistemu. Ravno zato so bolj primerni za analiziranje komunikacijskih protokolov. To pomeni, da se akcije prožijo ne samo, ko imajo zadostno število žetonov v pogoju, temveč ko imajo zadostno število ustreznih žetonov.

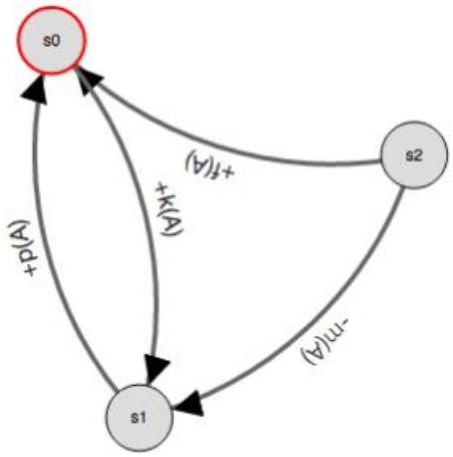
2.3 PGSS

Metoda PGSS je namenjena izrecno za analiziranje komunikacijskih protokolov. Graf, s katerim ponazorimo sistem, je sestavljen iz dveh ali več procesov, ki komunicirajo preko komunikacijskega kanala. Procesi v sistemu imajo čakalno vrsto poljubne dolžine, ki jo definiramo ob načrtovanju protokola. Primer na sliki 2.3 prikazuje dolžino čakalne vrste 1 za oba procesa.



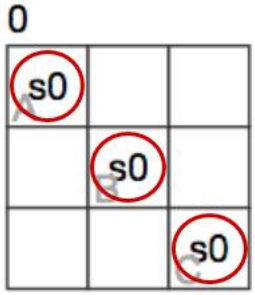
Slika 2.3: Sistem z dvema procesoma “A” in “B”

Vsak proces nato predstavimo s končnim avtomatom[10] stanj - FSM 2.4. Število stanj v procesu je poljubno, prav tako tudi povezav oziroma prehajanja med stanji. Vsak proces mora imeti samo eno začetno stanje.



Slika 2.4: Končni avtomat stanj - FSM s tremi stanji

Analiza protokola nam omogoča odkrivanje napak. Protokol analiziramo tako, da zgradimo drevo globalnih stanj. Vsako vozlišče vsebuje matriko velikosti števila procesov. Če imamo dva procesa, potem bo matrika velikosti 2 x 2. Elementi v glavni diagonali nam predstavljajo trenutno stanje procesov, ostali pa njihove čakalne vrste.

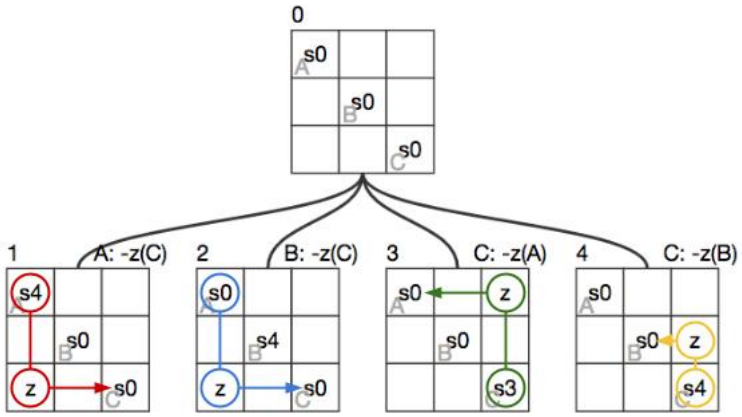


Slika 2.5: Začetno globalno stanje drevesa

Slika 2.5 je začetno globalno stanje drevesa, kjer ima protokol tri procese. Vsi procesi se nahajajo v stanju “s0”.

Drevo nato razvijamo na podlagi pošiljanja komunikacijskih sporočil med procesi. Vsako vozlišče drevesa označimo s številko, začetno globalno stanje ima številko 0. V desnem kotu matrike pa zapišemo anotacijo, ki nam predstavlja,

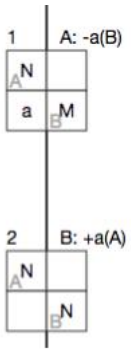
kateri proces komunicira s katerim in za kakšno vrsto komunikacije gre. Poznamo tri različne vrste komunikacije. Ob vsaki izmenjavi sporočil mora proces zamenjati svoje stanje.



Slika 2.6: Pošiljanje sporočil med procesi

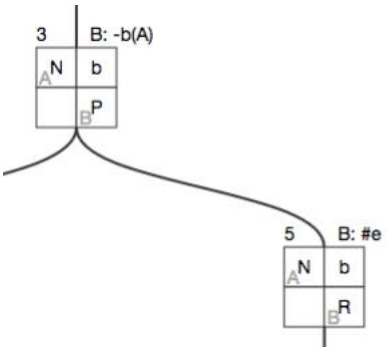
Na sliki 2.6 je predstavljeno pošiljanje sporočil med procesi. Slednje se označuje z znakom "-". Rdeča označba nam prikazuje, kako proces "A" pošlje sporočilo "z" procesu "C". Ta se postavi v čakalno vrsto procesa "C".

Naslednja slika 2.7 nam prikazuje sprejemanje sporočila. Sprejemanje označimo z znakom "+". Prikazano je, kako proces "B" sprejme komunikacijsko sporočilo "a" iz svoje čakalne vrste procesa A, saj ima vsak proces toliko čakalnih vrst, kolikor je ostalih procesov.



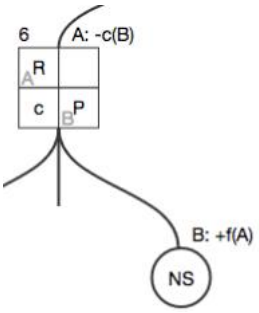
Slika 2.7: Sprejemanje komunikacijskega sporočila

Poznamo pa tudi lokalna sporočila, sporočila znotraj procesa. Označujemo jih z znakom "#". Predstavljajo akcije znotraj procesa, zato ta prav tako spremeni svoje stanje. Slika 2.8 prikazuje pošiljanje lokalnega sporočila v procesu "B", ki spremeni stanje iz "P" v "R".



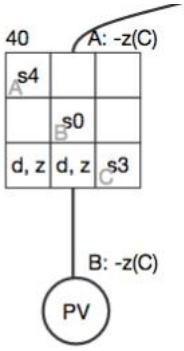
Slika 2.8: Pošiljanje lokalnih sporočil

Po končani analizi dobimo zgenerirano drevo globalnih stanj drevesa, ki pa običajno vsebujejo tudi napake oziroma drugačna vozišča, kot smo jih spoznali do zdaj. Analiza komunikacijskega protokola nas lahko pripelje do treh vrst napak. Prva je nedefiniran sprejem - NS, druga polna čakalna vrsta - PV, tretja pa smrtni objem - SO.



Slika 2.9: Nedefiniran sprejem - NS

Na sliki 2.9 lahko vidimo, da je prišlo do NS, ko je proces "B" želel sprejeti sporočilo "f" od procesa "A", vendar pa v njegovi čakalni vrsti ni bilo tega sporočila, temveč sporočilo "c". V primeru, da želi proces sprejeti sporočilo, čakalna vrsta pa je prazna, to ne štejemo kot NS, ampak drevesa ne razvijemo.



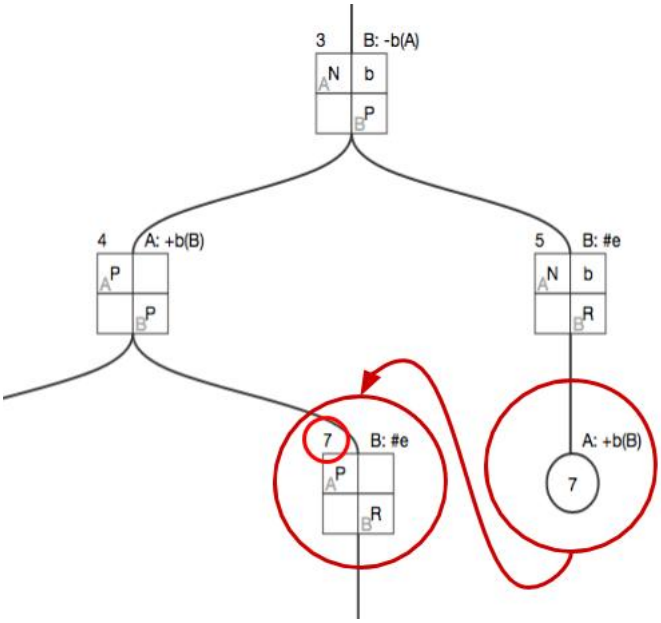
Slika 2.10: Polna čakalna vrsta - PV

Zgornja slika 2.10 prikazuje napako PV. Do napake pride, ko želi proces poslati sporočilo drugemu procesu, ta pa ima že polno čakalno vrsto.

Zadnji tip napake, smrtni objem, pa se zgodi, ko so vsi procesi pripravljeni sprejemati, vendar v sistemu nimamo nobenega procesa, ki pošilja sporočila drugim procesom, čakalne vrste pa so prazne. Tak sistem se ustavi in razvijanje vozlišč ni več mogoče.

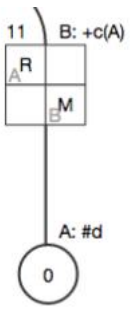
Obstaja pa tudi četrti tip vozlišča, in sicer za označevanje ciklov. Cikle, ki se dogajajo znotraj procesa in ne kažejo na začetno stanje, štejemo med napake. Predstavljajo mrtve zanke sistema.

Na sliki 2.11 je prikazan cikel. Ko proces “A” sprejme sporočilo “b” procesa “B” preide v isto stanje kot vozlišče številka 7.



Slika 2.11: Povezava na vmesno vozlišče številka 7

Vendar pa niso vsi cikli napake. Večina komunikacijskih protokolov teče v neskončni zanki, s tem omogočimo lažjo ponovno vzpostavitev komunikacije. Neskončna zanka lahko pomeni, da nek proces vseskozi posluša, ali bi želel kdo komunicirati z njim in si tako izmenjati sporočila. Te neskončne zanke se v grafu vidijo kot cikel z začetnim vozliščem. Naslednja slika 2.12 predstavlja ravno to.



Slika 2.12: Povezava na začetno vozlišče

2.4 ASN.1

Abstract Syntax Notation One[6] predstavlja standard za opisovanje oziroma definiranje strukture, kodiranja, pošiljanja in dekodiranja podatkov v telekomunikacijskih omrežjih. Osnovna pravila omogočajo predstavljanje z objekti, ki so med seboj neodvisni, niso povezani kot pri prejšnjih primerih, kjer uporabljamo končne avtomate oziroma grafe. Preproste anotacije omogočajo, da avtomatiziramo opravila za validiranje, tako je mogoče določena pravila zapisati v objekte in jih večkrat uporabiti.

ASN.1 je združeni standard[7] Mednarodne organizacije za standardizacijo (ISO), Mednarodne elektrotehniške komisije (IEC) in Mednarodne telekomunikacijske unije telekomunikacijskih standardov ITU-T. Zasnovan je bil leta 1984 kot del drugega standarda, vendar se je kasneje odcepil in leta 1988 je nastal nov standard X.208. V širšo uporabo je prišel leta 1995 in se vse do leta 2008 še izpopolnjeval. Vedno pa je ohranjal kompatibilnost med verzijami, zato je njegova uporaba še vedno precej pogosta.

2.5 Ugotovitve

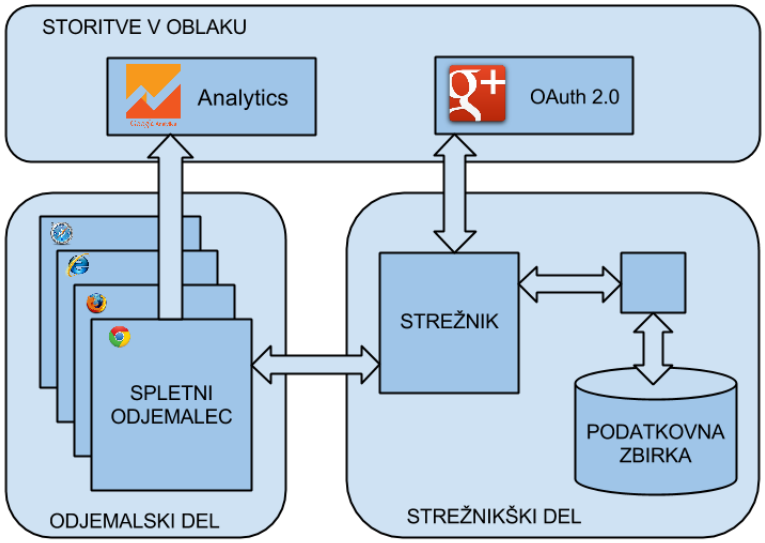
Do zdaj smo bolj podrobno spoznali tri različne načine analiziranja oziroma testiranja komunikacijskih protokolov, ki so najbolj razširjeni. Vsak izmed njih ima svoje prednosti. Vendar se lahko strinjamo, da je metoda PGSS najbolj učinkovita v odvisnosti od kompleksnosti metode. Prav zaradi preprostosti in ustrezne količine povratnih informacij sem se odločil za implementacijo ravno te metode. V nadaljevanju bom predstavil, kako je potekala implementacija.

Poglavje 3

Arhitektura spletne aplikacije

3.1 Uvod

Kot je razvidno na spodnji sliki 3.1 je zasnova aplikacije sorazmerno kompleksna. V strežniškem delu se nahaja spletni strežnik, ki se povezuje s podatkovno zbirko, kjer se shranjujejo protokoli oziroma projekti, ki jih uporabnik načrtuje. Prav tako se tja shranjujejo registrirani uporabniki. Omogočena je tudi prijava z Google računom, kar pomeni, da se mora aplikacijski strežnik preko protokola OAuth2 povezovati z Googlovo istoimensko storitvijo za preverjanje pristnosti uporabnika. Na aplikacijski strežnik se povezujejo uporabniki iz različnih spletnih brskalnikov, tudi mobilnih telefonov, ki dostopajo do spletne aplikacije, ki je naložena nanj. Zaradi kontrole obiska spletne aplikacije sem dodal še povezovanje z Googlovo storitvijo Analytics. Storitev služi beleženju akcij, ki jih uporabniki izvajajo, kar zelo pripomore pri nadgradnji uporabniškega vmesnika, saj lahko akcije, ki so večkrat izvedene, postavimo v ospredje oziroma dodamo bližnjice in tako naredimo aplikacijo še bolj prijazno uporabnikom.



Slika 3.1: Arhitektura spletne aplikacije

3.2 Tehnologije

Implementacija spletne aplikacije je zasnovana z najnovejšimi in najpopularnejšimi tehnologijami. V strežniškem delu se uporablja aplikacijski strežnik NodeJs in ogrodje Express. Za podatkovno zbirko se uporablja MongoDB in vmesnik Mongoose. Odjemalski del je prav tako celotno implementiran z uporabo JavaScript. Uporabljeno ogrodje za poslovno logiko je AngularJs, za izgled pa Bootstrap. Vse skupaj je združeno v ogrodje MEANJS, ki ob inicializaciji že vsebuje povezovanje s podatkovno bazo, kreiranje in prijavo uporabnikov ter druge konfiguracijske nastavitve.

3.2.1 Node.js

Node.js je aplikacijski strežnik, na katerem tečejo spletne aplikacije, napisane v programskem jeziku JavaScript. Podpirajo ga vsi operacijski sistemi, prav tako pa je enostaven za uporabo. Ob namestitvi se namesti tudi programček “npm” - (node package manager) za hitro pridobivanje oziroma nameščanje knjižnic.

Node.js je leta 2009 ustvaril Ryan Dahl skupaj z drugimi programerji podjetja

Joynet v San Franciscu. Že ob predstavitvi je požel bujen aplavz, do danes pa ga je samo še upravičil. Ker gre predvsem za novo rešitev, gre tudi razvoj zelo hitro, poleg tega pa je tudi odprtokoden kar pomeni, da lahko vsak prispeva svoj del napredka. Odprtokodnost projekta pomeni brezplačno uporabo tako v razvojnem kot tudi produkcijskem okolju.

3.2.2 MongoDB

MongoDB je NoSQL podatkovna zbirka. Gre za dokumentno orientirano bazo podatkov, v katero se shranjujejo JavaScript objekti. Prav zaradi tega je odlična izbira, saj se JavaScript objekti uporabljajo tako na odjemalčevi strani kot tudi na strežniški. To nam omogoča enostavno prenašanje in shranjevanje, saj ne potrebujemo nobenega pretvarjanja med podatkovnimi strukturami.

Prva verzija podatkovne zbirke je bila objavljena že leta 2007. Razvilo jo je podjetje MongoDB Inc. Tudi ta projekt je odprtokoden in se hitro razvija. Podpira zelo napredno iskanje in indeksiranje atributov vseh tipov. Omogoča hiter odziv tudi pri zelo veliki količini podatkov in je zaradi tega zelo priljubljena tudi pri velikih podjetjih kot so npr. Yahoo, Amazon itd.

3.2.3 AngularJS

AngularJS je MVC ogrodje na odjemalski strani in omogoča sistematično kodiranje. Njegova največja prednost je dvosmerno osveževanje dokumenta (Two-way Data Binding). Ob vsaki spremembi objekta v JavaScriptu se njena spremenjena vrednost vidi tudi v HTML dokumentu. Tako omogoča enostavno grajenje uporabniškega vmesnika. Popularen pa je tudi zaradi hitrega delovanja, saj se stran ne nalaga več v celoti, ampak se deli in podatki nalagajo preko asinhronih (Ajax) klicev.

Ogrodje je leta 2009 razvilo podjetje Google in ga izdalo kot odprtokodno rešitev, še vedno pa ima tudi danes največjo vlogo pri razvijanju novih različic.

3.2.4 D3.js

D3.js je JavaScript knjižnica, ki prav tako omogoča manipulacijo HTML dokumenta. Uporablja se za vizualizacijo podatkov oziroma gradnjo SVG elementa, ki

je vektorska slika v XML dokumentu. Pri souporabi z AngularJS je treba narediti nov modul, kot je opisano v knjigi [4].

3.2.5 Bootstrap

Bootstrap je CSS ogrodje, ki poenostavi oblikovanje uporabniškega vmesnika. Omogoča grajenje vmesnikov, ki so primerni tudi za mobilne naprave (responsive design). V ogrodje so že vključene standardne komponente: gumbi, vnosna polja, dvižni meniji itd.

3.2.6 Ostalo

Poleg naštetih knjižnic in ogrodij so v spletni aplikaciji uporabljene tudi manjše knjižnice za reševanje specifičnih primerov. Tako sem npr. na strežniškem delu uporabil knjižnico Async, ki služi asinhroni obdelavi in tudi paralelnemu izvajanju JavaScript funkcij. Uporabil sem jo pri shranjevanju komunikacijskega protokola, saj je treba shraniti precejšnjo količino podatkov v pravilnem vrstnem redu.

Vse uporabljene knjižnice so brezplačne za uporabo.

3.3 Implementacija

Kot sem že omenil, nam ogrodje MEANJS že ob namestitvi zgenerira kodo za upravljanje uporabnikov in osnovno postavitev strani. Tekom razvoja sem spremenil poslovno logiko dodajanja uporabnikov. Odstranil sem registracijo uporabnikov, saj menim, da bi to zmanjševalo uporabo aplikacije. Tako sem raje izbral prijavo z OAuth2 Google storitvijo, saj bo imela večina mojih uporabnikov Google račune. Vsi prijavljeni dobijo vlogo klasičnega uporabnika. Ustvaril sem tudi vlogo administratorja, ki lahko poleg klasičnih funkcionalnosti dodaja nove uporabnike oziroma briše in prepoveduje vloge ostalim uporabnikom.

Brez prijave lahko uporabniki samo gledajo že kreirane protokole, ne morejo pa jih ustvarjati. Po uspešni prijavi lahko vsak uporabnik ustvari tudi svoj protokol.

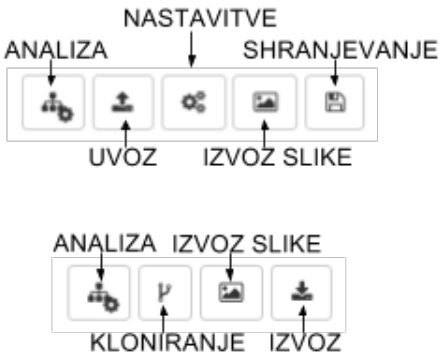
Kreiranje protokolov sem želel ustvariti zelo preprosto. Ob začetku kreiranja je treba vpisati ime protokola. Nato se nam odpre okno, kamor lahko vstavimo procese. Le-te enostavno dodamo tako, da kliknemo na gumb za dodajanje vozlišč, ki

je prikazan na naslednji sliki 3.2. Če želimo vozlišče izbrisati, ga s klikom označimo in pritisnemo gumb za brisanje vozlišča. Za povezovanje vozlišč uporabimo skrajno desni gumb, in sicer tako, da označimo vozlišče, kliknemo gumb za dodajanje, nato izberemo še ciljno vozlišče in ponovno pritisnemo gumb za dodajanje povezav.

Slika 3.2: Orodna vrstica za posamičen graf

Za malo večje sisteme sem omogočil tudi poljubno postavitev vozlišč. Privzeto se vozlišča ob dodajanju približajo sredini ustvarjalne površine, saj je tam gravitacijsko središče. Problem nastane, ko imamo veliko število elementov in se želijo vsi postaviti blizu središča, kar je za uporabnika nepregledno in nerazumljivo. Zato sem v ta namen dodal gumb za izklopitev gravitacije (skrajno levo na sliki 3.2).

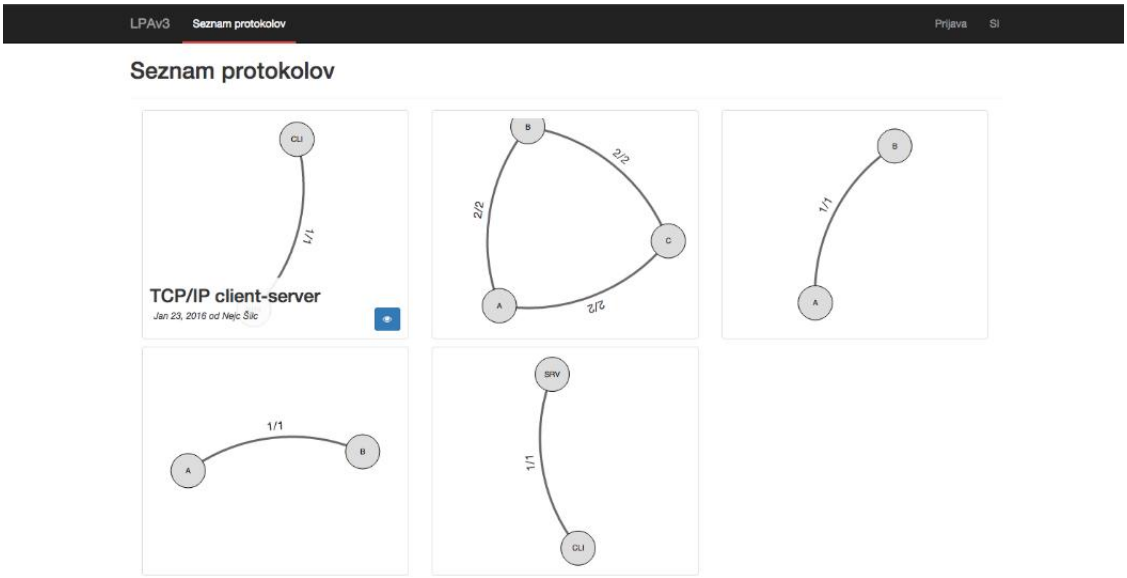
Prav tako so do zdaj omenjene akcije primerne pri ustvarjanju končnih avtomatov posameznega procesa. Za nastavljanje posameznih lastnosti procesa, povezave ali stanja je potreben dvoklik na element, kar povzroči odpiranje menija z nastavitvami.
V grafičnem oknu lahko zasledimo še eno orodno vrstico, in sicer namenjeno za akcije nad celim protokolom oziroma sistemom. Kot lahko vidimo na sliki 3.3, imamo gumbe za analizo, uvoz protokola v Json formatu, nastavitve, izvoz slike trenutno prikazanega grafa in gumb za shranjevanje protokola v podatkovno zbirko. Če pa protokola ne ustvarjamo oziroma ogledujemo že shranjeni protokol, pa imamo še možnosti kloniranja in izvoza protokola v Json formatu.



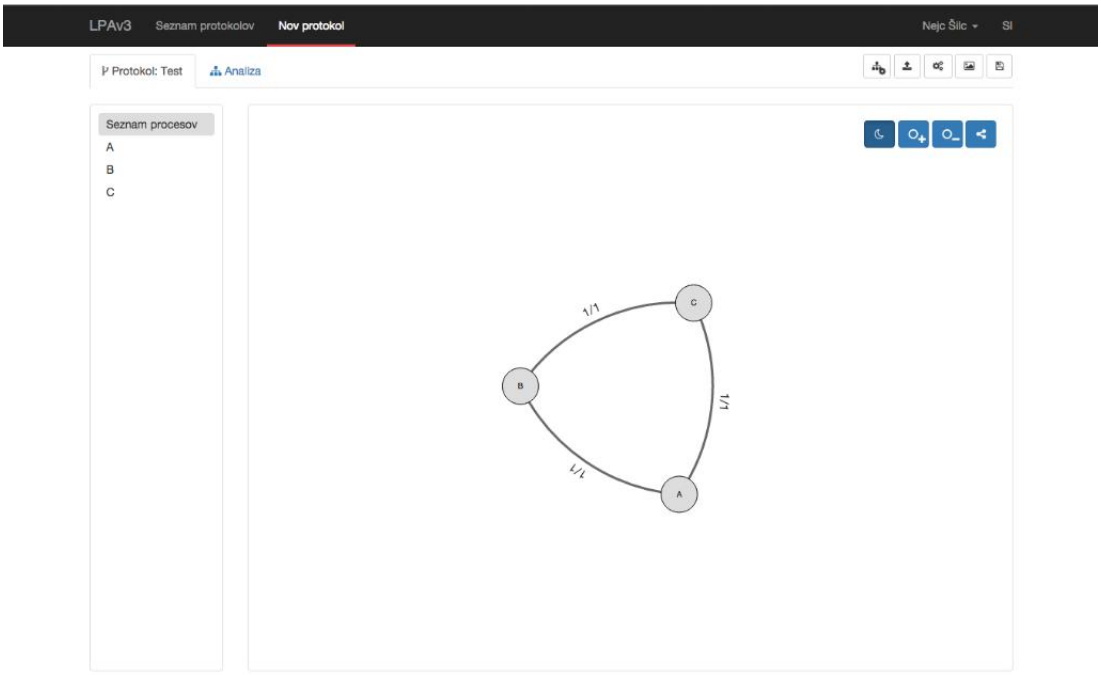
Slika 3.3: Orodni vrstici za protokol pri kreiranju novega ter ogledu shranjenega

3.3.1 Uporabniški vmesnik

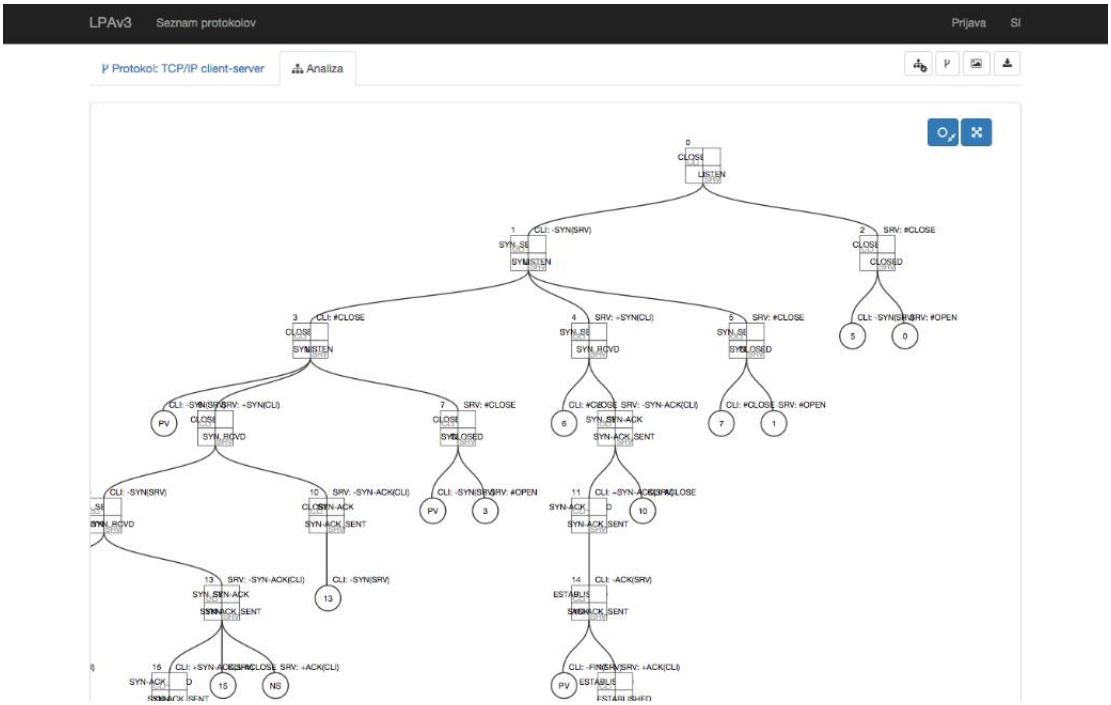
Slika 3.4: Domača stran spletne aplikacije



Slika 3.5: Seznam shranjenih protokolov



Slika 3.6: Kreiranje novega protokola



Slika 3.7: Analiza protokola pa metodi PGSS

Poglavje 4

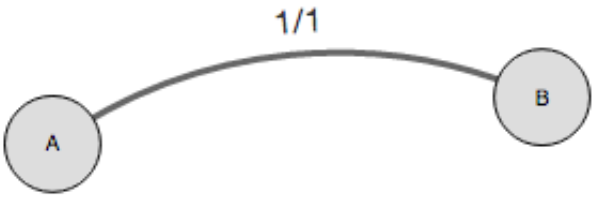
Testiranje in uporaba

4.1 Uvod

Zelo pomemben del ustvarjanja aplikacij je tudi testiranje, brez katerega praktično ne gre. Aplikacija LPA se pojavlja že dolgo. Z njo so se ustvarili številni primeri, ki so bili preizkušeni tudi ročno. Zato je bila odločitev, da testiranje izvedem z že znanimi in rešenimi protokoli, povsem logična. V sklopu predmeta Komunikacijski protokoli v 3. letniku Fakultete za računalništvo in informatiko smo spoznali preprost protokol, ki je predstavljen v naslednjem poglavju. Vendar moj cilj ni bil samo analiziranje preprostih protokolov, ki zahteva samo malo procesorske moči in je razvit graf lahko prikazati na majhnem zaslonu, temveč omogočiti uporabnikom tudi analiziranje kompleksnih komunikacijskih protokolov. Za primer sem izbral zelo znani protokol TCP, ki vsebuje 16 stanj ter po zagnani analizi zgenerira drevo s 360 vozlišči. Samo tako sem lahko preizkusil tudi uporabniško izkušnjo, kako pregledovati zelo veliko zgenerirano drevo.

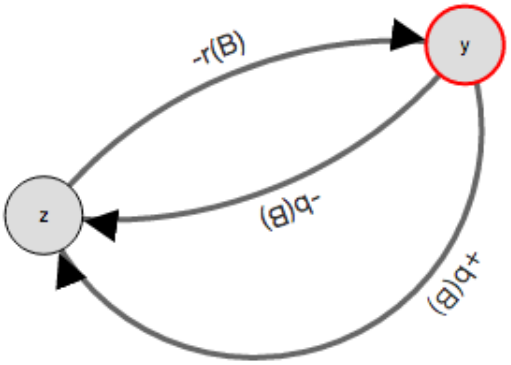
4.2 Preprost primer

Kot smo že omenili, komunikacijski protokol predstavimo z grafom. Na sliki 4.1 je preprost primer komunikacijskega protokola z dvema procesoma, “A” in “B”. Med njima je komunikacijska pot, po kateri se prenašajo sporočila iz procesa “A” v proces “B” in obratno. Oba procesa imata dolžino čakalne vrste ena.



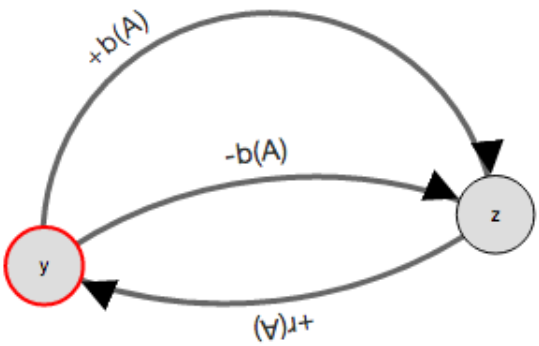
Slika 4.1: Preprosti primer komunikacijskega protokola

Proces “A” je na sliki 4.2 predstavljen s končnim avtomatom. In sicer vsebuje dva stanja, “z” in “y”. Stanje “y” je začetno stanje, iz katerega lahko v stanje “z” pridemo tako, da sprejmemo sporočilo “b” od procesa “B” ali pa sprejmemo sporočilo “b” od procesa “B”. Ko preidemo v stanje “z”, lahko pot nadaljujemo samo nazaj v začetno stanje in to le tako, da pošljemo sporočilo “r” procesu “B”.



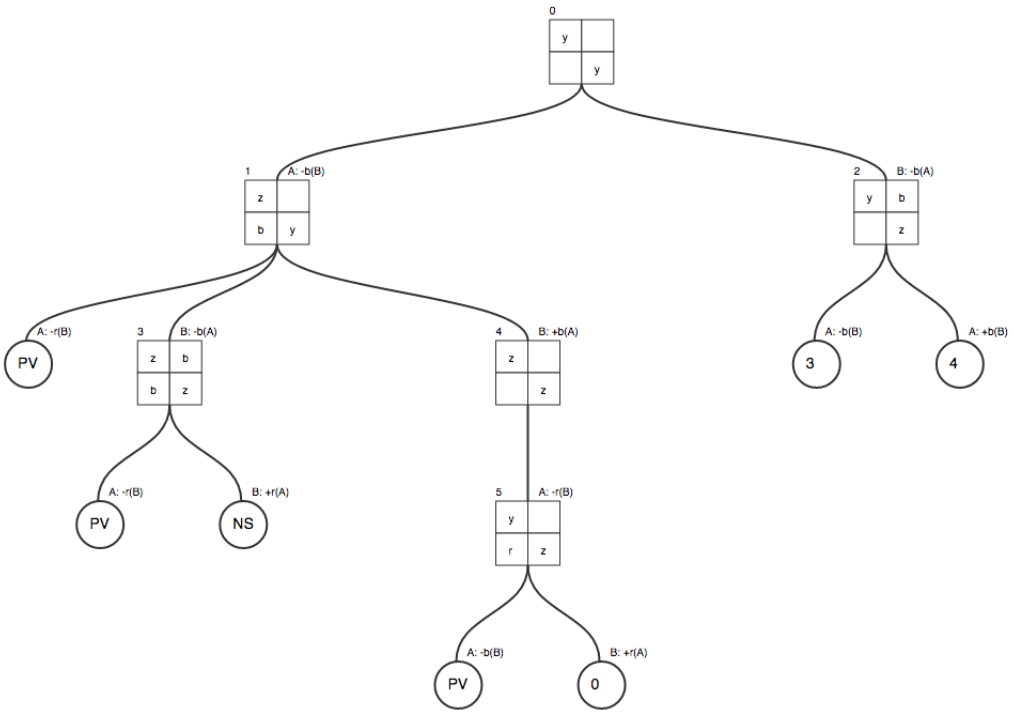
Slika 4.2: Proces “A”, predstavljen s končnim avtomatom

Proces “B” je podoben procesu “A”. Na sliki 4.3 lahko vidimo, da imamo prav tako dve stanji, “y” in “z”, pri čemer je tudi tukaj “y” začetno stanje. Iz začetnega stanja lahko pridemo v stanje “z”, samo če pošljemo ali sprejmemo sporočilo “b” procesa “A”. Iz stanja “z” se lahko vrnemo samo v primeru, ko sprejmemo sporočilo “r” procesa “A”.



Slika 4.3: Proces “B”, predstavljen s končnim avtomatom

Predstavljen protokol lahko zdaj analiziramo. Poslovna logika aplikacije nam zgenerira spodaj podano drevo 4.4. Vozlišča drevesa so matrike 2 x 2, saj imamo v protokolu dva procesa in dve čakalni vrsti.



Slika 4.4: Analiza preprostega komunikacijskega protokola

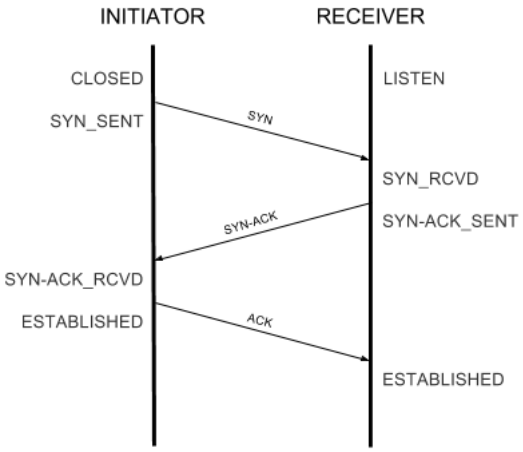
Iz podanega drevesa 4.4 lahko razberemo, da se že v tretjem nivoju pojavi polna vrsta, in sicer ko želi proces “A” poslati sporočilo “r” procesu “B”. V tem nivoju pa se dvakrat pojavi tudi cikel. V četrtem nivoju spet prihaja do polne vrste, prav tako pride do nedefiniranega sprejema, ko želi proces “B” sprejeti sporočilo “r” od procesa “A”. Drevo se razvije do petega nivoja, saj se nato vsa stanja postavijo v začetna, prav tako postanejo tudi vse čakalne vrste v procesih prazne. Pojavi pa se še ena polna vrsta, ko želi proces “A” poslati sporočilo “b” procesu “B”.

4.3 Protokol TCP

Za zahtevnejši primer protokola s stanji sem vzel protokol TCP. Gre za povezovalni protokol navadno med strežnikom in odjemalcem. Omogoča zanesljivo komunikacijo, saj se za vsako poslano sporočilo pričakuje potrditveno sporočilo. V primeru napake ali nedostavljenega potrditvenega sporočila se podatki pošljejo

ponovno. Zagotavlja tudi, da se podatki prenesejo v pravilnem vrstnem redu, kar doseže tako, da vsako poslano sporočilo označi z zaporedno številko. Optimizacije protokola so skozi čas omogočile tudi pošiljanje več paketov zaporedoma oziroma potrjevanje več prejetih sporočil z enim samim potrditvenim sporočilom.

Kot sem že omenil, je protokol TCP povezovalni protokol. To pomeni, da se mora med odjemalcem in strežnikom najprej vzpostaviti povezava. Slika 4.5 prikazuje primer vzpostavljanja povezave.

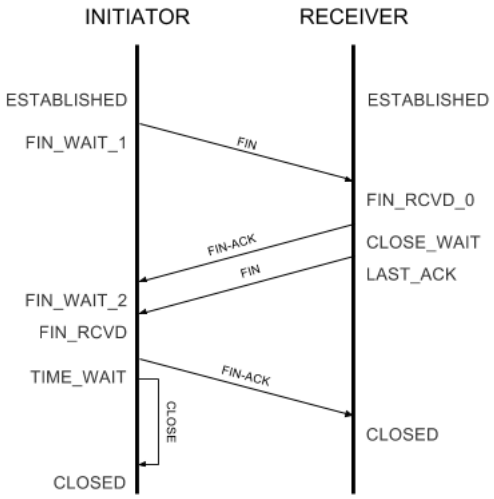


Slika 4.5: Diagram vzpostavljanja povezave pri protokolu TCP

Kot lahko vidimo na sliki 4.5, mora biti strežnik aktiven, če želimo, da se povezava vzpostavi. Nato mora odjemalec poslati zahtevo za vzpostavitev komunikacije, pošlje sporočilo "SYN". Ko strežnik uspešno sprejme sporočilo, se odzove s potrditvijo prejema, in sicer s sporočilom "SYN-ACK". Odjemalec se po prejemu tega sporočila odzove s sporočilom "ACK", ki je potrditveno sporočilo, da je uspešno prejel sporočilo s strani strežnika. Komunikacija je tako vzpostavljena in pravi prenos podatkov se lahko prične. Taka vzpostavitev povezave se imenuje trismerno rokovanje, saj je za uspešno vzpostavitev potrebno pošiljanje treh sporočil.

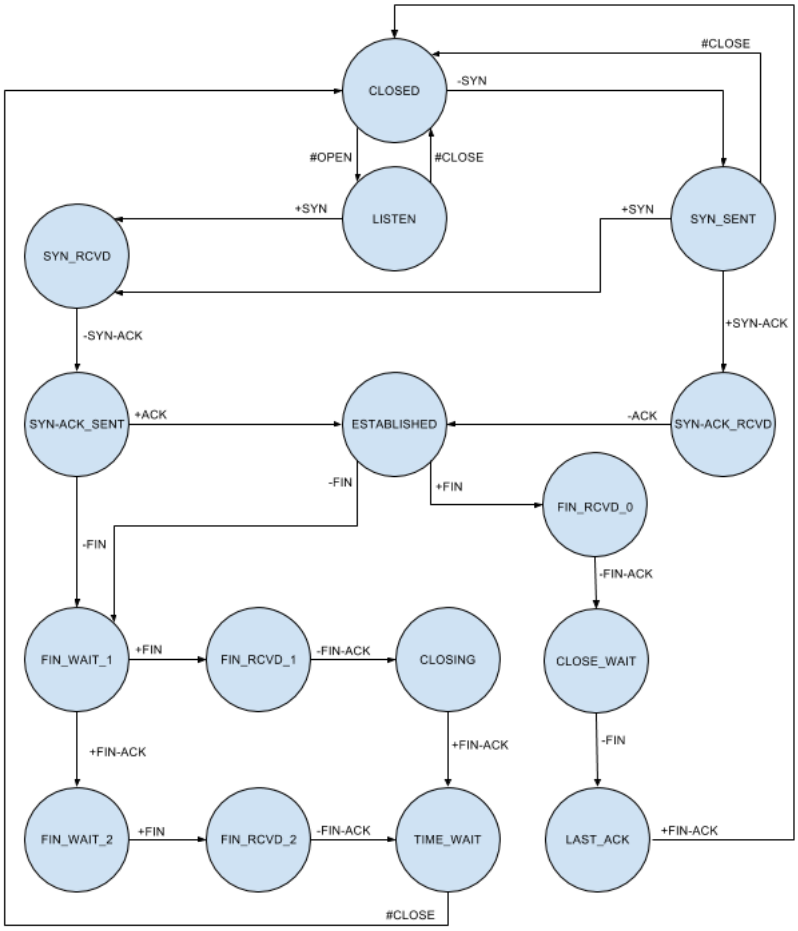
Na sliki 4.6 je prikazan postopek zaprtja povezave. Uporablja se štirismerno rokovanje, in sicer mora odjemalec najprej poslati sporočilo "FIN", ki sporoča, da želi prekiniti povezavo. Strežnik nato pošlje potrditveno sporočilo "FIN-ACK", ki potrdi sprejem sporočila za prekinitev povezave, in sporočilo "FIN", ki pomeni,

da tudi on želi zapreti povezavo. Odjemalec se po uspešnem prejemu odzove s "FIN-ACK", nato pa navadno počaka 240 sekund in tudi sam zapre povezavo.



Slika 4.6: Diagram zapiranja povezave pri protokolu TCP

Na zgornjih slikah 4.5 in 4.6 so označena tudi stanja, v katerih se odjemalec oziroma strežnik nahaja. Ob prejemu ali pošiljanju sporočila se stanje spremeni. Protokol TCP ima 16 stanj. Prehodi med njimi so predstavljeni na naslednji sliki 4.7.



Slika 4.7: Protokol TCP - končni avtomat stanj

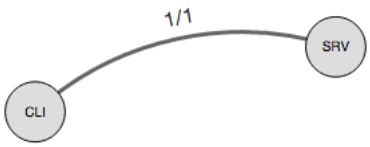
Začetno stanje vsakega procesa je “CLOSED”. V tem stanju proces lahko prične z vzpostavljanjem povezave ali pa preide v stanje “LISTEN” in tako čaka na zahtevo za vzpostavitev komunikacije. V primeru, da želi vzpostaviti povezavo, v stanju “CLOSE” pošlje sporočilo “SYN” in preide v stanje “SYN_SENT”. Temu pravimo aktivno odpiranje povezave. Pasivno odpiranje povezave pa je v primeru, ko iz stanja “LISTEN” sprejme sporočilo “SYN” in preide v stanje “SYN_RCVD”. Ko je v tem stanju, pošlje sporočilo “SYN-ACK”, ki je potrditveno sporočilo, da je prejel zahtevo za vzpostavitev komunikacije, in preide v stanje “SYN-ACK_SENT”. Nato mora samo še počakati sporočilo “ACK”, ki ga dobi, ko drugi proces po odpiranju aktivne povezave sprejme sporočilo “SYN-

ACK" in preide v stanje "SYN-ACK_RCVD" ter pošlje sporočilo "ACK". Tako lahko oba procesa preideta v stanje "ESTABLISHED", kar pomeni, da je povezava vzpostavljena in da se prenos podatkov lahko prične.

Zapiranje povezave se začne, ko proces v stanju "ESTABLISHED" pošlje sporočilo "FIN". Sam preide v stanje "FIN_WAIT_1", proces na drugi strani pa po prejemu sporočila preide v stanje "FIN_RCVD_0". Po prejemu sporočila "FIN" pošlje potrditveno sporočilo "FIN-ACK" in preide v stanje "CLOSE_WAIT". Za tem pošlje še sporočilo "FIN", ki pomeni, da želi tudi on končati povezavo ter preide v stanje "LAST_ACK", kjer čaka, da prejme sporočilo "FIN-ACK". Proces na drugi strani po prejemu sporočila "FIN" preide v stanje "FIN_RCVD_2", pošlje sporočilo "FIN-ACK" in spremeni stanje v "TIME_WAIT", kjer počaka določen čas, nato pa preide v stanje "CLOSED". Prav tako po prejemu sporočila "FIN-ACK" v omenjeno stanje preide tudi proces, ki je sprožil zapiranje povezave.

To je običajno odpiranje in zapiranje povezave. Omogočeno pa je tudi zapiranje povezave iz stanja "SYN-ACK_SENT". To se zgodi, ko proces pošlje zahtevo za odpiranje povezave, vendar jo nato zapre, še preden se povezava vzpostavi. Poskrbljeno je tudi za situacijo, ko želita oba procesa v komunikaciji naenkrat zapreti povezavo. Tisti proces, ki prvi pošlje sporočilo "FIN" po prejemu istoimenskega sporočila, preide v stanje "FIN_RCVD_1", pošlje sporočilo "FIN-ACK", spremeni stanje v "CLOSING" in počaka na "FIN-ACK" drugega procesa. Po prejemu sporočila preide v že omenjeno stanje "TIME_WAIT".

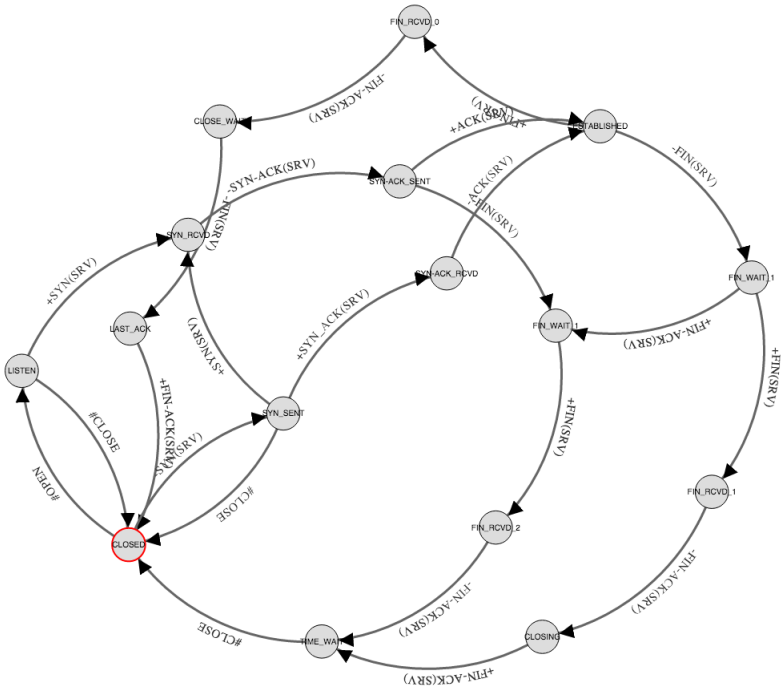
Z uporabo aplikacije LPA sem uspel realizirati sistem, kjer imamo dva procesa, ki komunicirata po protokolu TCP. Kot vidimo na sliki 4.8, sem ju poimenoval "SRV" in "CLI".



Slika 4.8: Procesi protokola TCP - odejmalec - strežnik

Na naslednji sliki je prikazan končni avtomat stanj za proces "CLI". FSM

za proces “SRV” je isti, oba pa se ujemata z zgoraj opisanimi stanji protokola TPC/IP.



Slika 4.9: Končni avtomat stanj za proces “CLI”

Analiza protokola nas pripelje do ogromnega globalnega drevesa stanj sistema, ki je dodan kot priloga. Drevo je zelo veliko predvsem zato, ker nisem upošteval prioritet, pa tudi zato, ker so upoštevane akcije, ki se v praktični uporabi ne zgodijo. V sistemu se pojavljajo tudi polne vrste. Ena se pojavi, ko proces “CLI” pošlje zahtevo “SYN”, sam nato preide v stanje “CLOSED”, od koder spet lahko pošlje zahtevo procesu “SRV”, vendar je proces “SRV” v stanju “CLOSED” in tako ne sprejema sporočil procesa “CLI”. Prihaja tudi do nedefiniranih sprejemov in veliko ciklov. Vse te napake pa so le posledica implementacije obeh procesov z istim končnim avtomatom. V praksi je upoštevano, da proces “SRV” ne bi smel pošiljati zahtevkov “SYN” procesu “CLI”, saj med drugim niti ne pozna njegovega naslova. Prav tako bi procesu “CLI” onemogočili poslušanje zahtevkov “SYN”.

Poglavje 5

Sklepne ugotovitve

5.1 Rezultat

Izdana verzija aplikacije LPAv3 ob predstavitvi diplomskega dela je velik napredek vseh dosedanjih verzij aplikacije LPA. Omogočene so vse do zdaj znane funkcionalnosti, dodane pa so še nekatere druge, ki smo jih v dosedanjih zelo pogrešali. Največjo prednost ima v zelo enostavnem in učinkovitem pregledovanju drevesa globalnih stanj sistema. Takoj za tem pa bi izpostavil enostavno dostopnost do aplikacije. Do nje lahko dostopamo brez predhodne namestitve, saj je objavljena na spletu. Enostavno je tudi deljenje načrtovanih protokolov, saj se ti shranjujejo v skupno podatkovno zbirko.

Z izdelavo oziroma s končnim izdelkom sem zadovoljen, vendar pa bi lahko nekatere stvari še izboljšal. Izvorna koda aplikacije je objavljena na GitHubu (https://github.com/nejcsilc/lpa) in je dostopna vsem. Omogočeno je sodelovanje na projektu in pa prijavljanje napak, ki jih bom poskušal reševati tudi po predstavitvi oziroma zaključku šolanja. Na tem mestu bi poudaril svojo največjo napako, in sicer pisanje testov, ki služijo preverjanju pravilnosti delovanja. Tekom razvoja nisem uspel pripraviti avtomatičnega procesa, ki bi preverjal ali algoritmi za analiziranje delujejo pravilno. To pomeni, da je treba ob vsaki spremembi ali nadgradnji aplikacije preveriti kar veliko robnih primerov, če želimo, da aplikacija zagotovo deluje v celoti.

Prav tako mi je žal, ker nisem na strežniški strani uporabil nove, 6. verzije JavaScripta in se tako naučil programerjem bolj razumljive sintakse programskega jezika.

Čeprav sem se z izdelavo zelo trudil in sem poskušal implementirati čim več zanimivih funkcionalnosti, vidim še nekaj zelo uporabnih dodatkov. Zagotovo bi bilo treba izboljšati dodajanje povezav med vozlišči, in sicer samo s potegom miške od enega do drugega vozlišča, ali pa majhno okno pri analizi, kjer bi bil trenuten graf v nekaj merilih večji in kjer bi bilo označeno, kateri del grafa trenutno pregledujemo. Zelo uporabna bi bila funkcionalnost razveljavi-ponovi, ki bi razveljavila zadnjo potezo pri načrtovanju procesov ali končnih avtomatov. Uporabno bi bilo tudi sprotno shranjevanje, ki bi ob vsaki spremembi na načrtovalni površini protokol shranilo v podatkovno zbirko. Zagotovo pa je teh izboljšav še veliko in jih bom skušal dodajati postopoma oziroma se mogoče po zaključku mojega šolanja najde nekdo, ki bi želel sodelovati in nadgraditi moje začeto delo.

Trenutno sem aplikacijo namestil v brezplačno oblačno storitev (PaaS) Heroku, kjer je dosegljiva na naslovu http://lpa3.herokuapp.com. Prav tako sem zakupil domeno lp3.site (http://lpa3.site) in omogočil, da je aplikacija dosegljiva tudi na naslovu, ki je neodvisen od mesta namestitve.

5.2 Pričakovanja in uporaba

Vse dosedanje različice aplikacije LPA so se uporabljale tudi v okviru študija na Fakulteti za računalništvo in informatiko v 3. letniku pri predmetu Komunikacijski protokoli. Glavno funkcionalnost aplikacije uporabljajo vsi študentje tega predmeta, saj z njo lahko preverijo razvita drevesa globalnih stanj, ki jih razvijajo ročno. Prav tako je temu namenjena tudi moja aplikacija. Študentje se bodo v aplikacijo enostavno prijavili z istim uporabniškem računom, kot ga uporabljajo za dostop do Studisa ali spletne učilnice. Po uspešni prijavi bodo lahko začeli načrtovati in testirati oziroma analizirati nove komunikacijske protokole ter rezultate primerjali z ročnimi rešitvami. Zaradi shranjevanja protokolov v skupno podatkovno zbirko pa bodo nadzor nad njimi imeli tudi profesorji in asistenti. Ti bodo lahko preverjali, ali je posamezen študent načrtoval kakšen protokol in v

primeru, da ga ne bo, enostavno utemeljili neuspeh pri pisnem izpitu.

5.3 Vtisi

Izdelava diplomskega dela je bila zelo zahtevna. Zadal sem si obsežno delo, a sem ga uspešno opravil. V največje veselje mi je bila izdelava spletne aplikacije, saj me to področje zelo zanima in veseli. Na začetku je bilo potrebnega kar veliko premišljevanja, saj sem moral načrtovati vsako še tako majhno podrobnost, npr. katere tehnologije uporabiti, da bo zadeva delovala hitro in da ne bo potrebno kupovati licenc. Med izdelovanjem diplomskega dela sem se naučil veliko novih stvari oziroma utrdil do zdaj pridobljeno znanje. Vesel sem, ker sem uporabljal pretežno nove tehnologije in bom lahko svoj praktični izdelek predstavljal kot reference pri iskanju zaposlitve, saj so povpraševanja po tovrstnih tehnologijah zelo velika.

Literatura

[1] C. Costea, D. Costea, V. Groza, B. Groza, and D. Costea. Petri net reduction. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 1527–1530, April 2007.

[2] A. Helmy, D. Estrin, and S. Gupta. Systematic testing of multicast routing protocols: analysis of forward and backward search techniques. In *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*, pages 590–597, 2000.

[3] D. Hoffman. The trace specification of communications protocols. *Computers, IEEE Transactions on*, C-34(12):1102–1113, Dec 1985.

[4] C. Körner. *Data Visualization with D3 and AngularJS*. Community experience distilled. Packt Publishing, 2015.

[5] Marco Ajmone Marsan, G. Balbo, Gianni Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[6] Nilotpal Mitra. Efficient encoding rules for asn.1-based protocols. *AT T Technical Journal*, 73(3):80–93, May 1994.

[7] D. Tantiprasut, J. Neil, and C. Farrell. Asn.1 protocol specification for use with arbitrary encoding schemes. *Networking, IEEE/ACM Transactions on*, 5(4):502–513, Aug 1997.

[8] A. Tejada, O.R. Gonzalez, and W.S. Gray. Stability analysis of stochastic hybrid jump linear systems using a markov kernel approach. *Automatic Control, IEEE Transactions on*, 58(12):3156–3168, Dec 2013.

[9] Tone Vidmar. *Analiza porazdeljenih sistemov*. Fakulteta za elektrotehniko in računalništvo, 1991, Ljubljana, Slovenija, 1st edition, 1991.

[10] M.M. Wu and M.C. Loui. Modeling robust asynchronous communication protocols with finite-state machines. *Communications, IEEE Transactions on*, 41(3):492–500, Mar 1993.